AD-A100 518 FOREIGN TECHNOLOGY DIV WRIGHT-PATTERSON AFB OH F/G 9/2
A SOFTWARE FOR CAD PHOTOMASK --- ZB-761,(U)
MAY 81 H XIAN-LONG
UNCLASSIFIED FTD-ID(RS)T-0018-81 NL

1 OF 1
AD
A100518

END
DATE
FILMED
7-81
DTIC

FTD-ID(RS)T-0018-81

FOREIGN TECHNOLOGY DIVISION

A SOFTWARE FOR CAD PHOTOMASK --- ZB-761

by

Hong Xian-long

Approved for public release;
distribution unlimited.

14 FTD-ID(RS)T-0018-81

EDITED TRANSLATION

FTD-ID(RS)T-0018-81 21 May 1981

MICROFICHE NR: FTD-81-C-000450

6 A SOFTWARE FOR CAD PHOTOMASK --- ZB-761

By: Hong/Xian-long

English pages: 27

Source: Journal of Qing Hua University, Vol. 19, Nr. 2, 1979, pp. 63-73

Country of origin: China
Translated by: SCITRAN
F33657-78-D-0619
Requester: FTD/TQTR
Approved for public release; distribution unlimited.

21 May 81

29

THIS TRANSLATION IS A RENDITION OF THE ORIGINAL FOREIGN TEXT WITHOUT ANY ANALYTICAL OR EDITORIAL COMMENT. STATEMENTS OR THEORIES ADVOCATED OR IMPLIED ARE THOSE OF THE SOURCE AND DO NOT NECESSARILY REFLECT THE POSITION OR OPINION OF THE FOREIGN TECHNOLOGY DIVISION.	PREPARED BY: TRANSLATION DIVISION FOREIGN TECHNOLOGY DIVISION WP-AFB, OHIO.

600

A SOFTWARE FOR CAD PHOTOMASK --- ZB-761

Hong Xian-long

Department of Electronic Engineering,

Qinhua University

Abstract

As a part of the LSI CAD, a software for CAD photomask --- ZB-761 was designed and put into use on a DJS-130 computer in 1976. Since then several dozens of IC photomasks have been made.

The input language of the software has a fairly high ability of describing IC patterns. Its data structure enables minicomputers such as DJS-130 (without the external storage) to handle a large amount of IC pattern data. The mask fabrication time using special equipments has been reduced extensively because of the optimization of the object program. Furthermore, the software possesses some ability of data checking and man-machine conversation.

This paper introduces in detail the characteristics of the input language of the software, data structure, transformation computation in compilation, and optimization of the object program, etc. It also proposes some idea about the automation in LSI pattern design.

Introduction

Computer Aided Design of Photomask is a technology of fabricating photomasks of Large Scale Integrated-circuits using computers and special mask-fabrication equipments. It is a part of LSI CAD.

CADP can be realized by writing a source program in the form of specially designed "CADP language" from a man designed IC pattern

sketch, inputing it into a computer by means of paper tape or keyboard. After the processing of the CADP language compiler, the computer produces a paper tape for program control which is then used to drive program controlled mask fabrication equipments to fabricate intermediate-step photomasks (these program controlled equipments may also be connected directly with the computer to fabricate photomasks). Up till now, the program controlled mask fabrication equipments are automatic mask cutting machines, pattern generators and electron beam mask fabrication equipments. It is more often to use the former two. Also, CADP may be realized by inputing pattern data and other necessary information directly into the computer by means of "man-machine conversation" pattern input equipments, if the availability of facilities allows, and processing these data with the computer. The block diagram of CADP is shown in Fig.1.

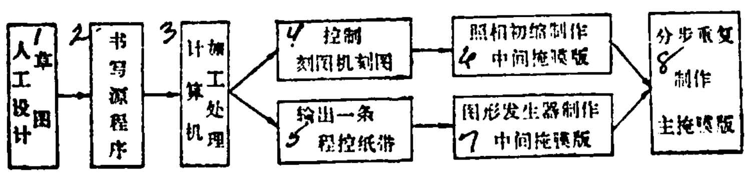


Fig.1 The block diagram of CADP

1 - Man designed IC pattern sketch

2 - Writing a source program

3 - Computer processing

4 - Control the mask cutting machine to operate

5 - Output a program controlled paper tape

6 - Fabricate the intermediate-step photomasks by means of miniature photographing

7 - Pattern generator fabricates the intermediate-step photomasks

8 - Step-by-step objective photomask fabrication

There are six parts in this paper: 1) An outline of ZB-761 input language; 2) Data structure of the software; 3) Transformation computation in compilation; 4) Optimization of the objective program; 5) Result checking and man-machine conversation; 6) Ideas and preparations of realizing automatic design.

I. AN OUTLINE AND SOME CHARACTERISTICS OF ZB-761 INPUT LANGUAGE

ZB-761 language is a specially designed language for IC photomask fabrication. There are more than thirty statements in this language which are divided into three types. The first type is "declaration type" which describes the technological specifications in fabrication and the working conditions of the pattern generator. The second one is "pattern description type" which describes the photomask patterns. The third one is "operation type" which drives various peripheral equipments to operate.

The construction of the source program of ZB-761 language can be shown with Fig.2.

The "subpattern statements" in Fig.2 are the ones that describe a rectangular and simple combinations of rectangulars. Also, the "transformation statements" describe translation, symmetrization and rotation of subpatterns. The subpattern statements are the most basic ones among all these statements since a rectanguler is the simplest unit subpattern in a digital circuit pattern.

- source program
  - declaration part
    - technological specification declaration statements
    - declaration statements for the upper platform of the pattern generator
    - unit circuit library declaration statements
  - pattern description part (region program segments and the entire pattern program segment)
    - beginning of the program segments
    - pattern description part in the program segments
      - subpattern statements
      - transformation statements
      - combination statements
      - combined subpattern declaration
      - call statements
      - layer piling statements
      - interconnection line statements
      - automatic design statements
      - other statements
    - end of the program segments
  - operation part
    - printout statements
    - punchout statements
    - draw statements
    - display statements

Fig.2 Source program structure of ZB-761 language

We will not explain these statements one-by one. Instead, we shall compare them with some standard programming language to introduce some characteristics of this language. For more detail please refer to [1].

Now we compare the language ZB-761 with FORTRAN.

(1) Block structure --- program segment

The key part of the ZB-761 language, that is, the pattern description part, is structured in blocks. It is comprised of an "entire pattern program segment" corresponding to a main program and several "region program segments" corresponding to subprograms.

In spite of the complexity that a digital circuit may have, it can always be decomposed into some subcircuits (e.g. gates,flip-flops, shift registers, etc.). The corresponding design pattern is also comprised of subpatterns corresponding to these subcircuits and the interconnection lines between them. Thus we may define these subcircuit patterns as "region program segments", and only use call statements in the main program to form a desired pattern. This can also be generalized to any frequently-appearing subpatterns.

The characteristics of the block structure enables each program segment to describe a subpattern independently. It also ensures the separation of writing, debugging and checking of the source program. Also, it is convenient for several people to write the source program for the same design pattern.

(2) Parameters and arguments --- definition, calling and transformation of program segments

In defining the region program segments, although a user writes the source program in accordance with the actual coordinates

in a design pattern, the region program segments after compiling do not represent the actual pattern. They only represent "floating" subpatterns with relative coordinates. The actual pattern is reconstructed by the arguments in the call statements. The arguments include the X, Y coordinates of the lower-left corner of the subpatterns and the transformation statements describing the geometrical transformation relation.

For example, Fig.3 shows a design pattern which includes three identical subpatterns with different positions and different orientation. Considering the same subpattern Q_1 in position (0,0), we define a region program segment named Q_1. The subpattern Q_1 may be single-layered or multi-layered. The actual subpatterns at (x_0, y_0), (x_1, y_1), (x_2, y_2) are realized by calling Q_1. This can be written in ZB-761 language as:

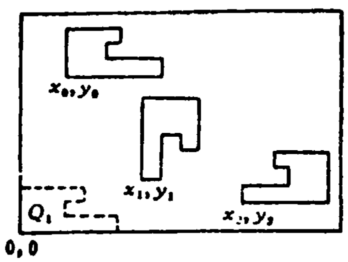


Fig.3

#DY Q_1↵
#PY x_0, y_0↵
#DY Q_1↵
#SZ x_1, y_1↵
#DY Q_1↵
#YD x_2, y_2↵

where "↵"s are the statement seperators, "#DY", "#PY", "#SZ", "#YD" are the statements for "call", "translate", "rotate 90° clockwise" and "symmetrize with respect to Y axis". The execution of the first

two statements results in a translated Q_1 pattern at (x_0, y_0). . The middle two statements result in a 90° clockwise rotated Q_1 pattern at (x_1, y_1). . Finally, the last two result in a Y-axis symmetric pattern of Q_1 at (x_2, y_2) . The three transformation statements functioned as arguments in calling the subprogram.

(3) Functional statements --- combined subpattern declaration

As the functional statements in FORTRAN programming language, can define a "combined subpattern" with "combined subpattern declaration" statements in case that the same single-layered combined subpattern is frequently referred to in a subprogram, then realize the actual pattern by the call statements and the geometrical transformation statements.

For example, we assume that the subpatterns in Fig.3 are single-layered combined subpatterns. In the subprogram we can first define a combined subpattern T_1 with "combined subpattern declaration" statements and then call it with the transformation statements as the arguments to produce three actual subpatterns. This can be written as:

#DY T_1 ↵
#PY x_0, y_0 ↵
#DY T_1 ↵
#SZ x_1, y_1 ↵
#DY T_1 ↵
#YD x_2, y_2 ↵

(4) Nested calling

The defining and the calling of the combined subpatterns and subprograms has extensively improved the ability of pattern

description. However, practical circuit patterns often require the function of nested calling. ZB-761 allows the nests of calling subprograms or combined subpatterns up to 10 while does not allow any kind of self-calling. For examples of nested calling, please refer to Fig.4 and Fig.5.

(5) "Subroutine" --- standard unit circuit library

Again as standard subroutines in FORTRAN, ZB-761 is equiped with a "standard unit circuit library". Frequently used basic circuit subpatterns are stored in some prescribed way in the library by means of preprogramming. During the fabrication, the unit circuit library is called and transformed geometrically to produce the desired subpatterns.

Since the unit circuit library is usually stored in the external storage, they must be declared by "unit circuit library declaration" statements in order to dump these unit circuit subpatterns into the internal storage for use.

The design of the unit circuit library not only simplifies the source program, reduces a large amount of work in reading coordinates, it also facilitates the further work such as pattern assembly and layout.

Other functions and characteristics of this language such as transformation of subpattern statements, additivity of transformation, successive use of transformation statements, and pattern combination which is similar to do loop will not be dealt with here.

II. DATA STRUCTURE

Any programming language must deal with the issue of data

structure. This problem becomes more important in CADP language since this language may be viewed as a program for processing graphic data in some sense.

There are usually a large number of rectangulars up to several dozen thousand or even several hundred thousand in an IC pattern. It will be very space and time consuming if the data of these rectangulars are stored directly in the computer. Besides, it will be very inconvenient to find, to convert, or to modify a particular one of them. Thus it is necessary to design a data structure to store these graphic data so as to save the internal storage and to make it easier to find, to convert, and to modify them.

A so called "intermediate result form" chain structure is designed for ZB-761 as the basic data structure. After compiling, the source program is stored in the internal storage as an intermediate result. When the output part is executed, the data in the intermediate result form are transformed into various output form required by peripheral equipments.

A nested IC pattern is shown in Fig.4. The entire pattern Z_1 is comprised of three regions QA,QB and QC. QA contains QA1 and QA2 (names with initial letters Z are entire pattern names, similarly, Q's denote region names and T's denote combined subpattern names). QB contains QA1 and QB1. QC contains QA2 and QB1. Each QA1, QA2 and QB1 contains TA, TB and TC. TA is comprised of TB and a subpattern, TC is comprised of TA and the subpattern, while TB is comprised of the subpattern only. The subpattern is the basic graphic cell. After compiling, the data form in the internal storage is a chain structure as shown in Fig.5.

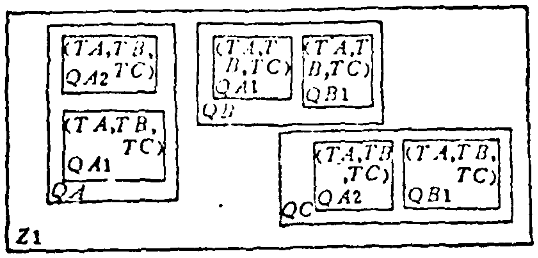
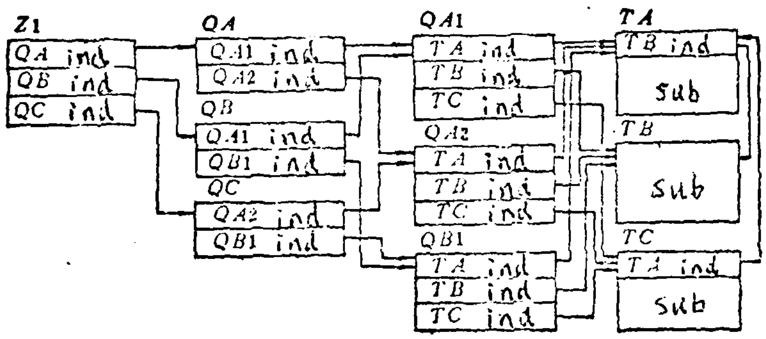


Fig.4 A nested pattern



ind. - indicator

sub. - subpattern

Fig.5 Data structure in the internal storage corresponding to Fig.4

The indicators Q and T in Fig.5 are comprised of six to ten internal memory cells for each. In case that the program segments or the combined subpatterns are only called and transformed, the indicators are comprised of six words as shown in Fig.6. This is called the "simple-indicator". In case that further combination is needed after those calling and transforming, the "multi-indicator"s comprised of seven to ten words as in Fig.7 are used. The transformation code in Fig.6 and Fig.7 represents the transformation

relation between the Q or T when they were defined and the Q or T after being called.

transformation code
entrance address of Q or T
length of Q or T
width of Q or T
X axis coordinate of the lower-left corner after being called
Y axis coordinate of the lower-left corner after being called

Fig.6

transformation code
entrance address of Q or T
length of Q or T
width of Q or T
X axis coordinate of the lower-left corner after being called
Y axis coordinate of the lower-left corner after being called
spacial interval of repetition in X direction
spacial interval of repetition in Y direction
times of repetition in X direction
times of repetition in Y direction

Fig.7

This structure obviously saves a large amount of the internal storage. It does not store any redundant parts. Also, due to the indicators, it is quite convenient to find each data block.

It is also convenient to modify the IC patterns with this data structure. For example, if we wish to add a QB1 in Z1, we only need to insert a QB1 indicator before the end code of Z1. Fig.8 shows the result of adding QB1 in Z1. Similarly, if we wish to delete a region QB, it suffices to substitute QB with a reject indicator. The result of deleting QB in the data block of Z1 is shown in Fig.9.

beginning of Z1
QA indicator
QB indicator
QC indicator
QB1 indicator
end of Z1

Fig.8

beginning of Z1
QA indicator
reject indicator
QC indicator
end of Z1

Fig.9

This data structure also makes the layout and assembly easier. In layout or assembly, we consider the object being assembled as an entity placed somewhere on the pattern regardless of the contents contained in the object. In order to assemble QA, QB1 and QA1 into Z2, for example, we construct a table shown in Fig.10, and fill in appropriate values for the transformation codes and lower-left corner coordinates of the QA, QB1 and QA1 indicators. If we want to change the position and the orientation of these regions in the pattern, we only need to change the above parameters.

Moreover, an "intermediate form of lines" and some other data forms have been designed for ZB-7 1, which give data structures of

the internal storage for manual interconnecting, lightpen interconnecting and automatic layout design.

beginning of Z2
QA indicator
QB1 indicator
QA1 indicator
end of Z2

Fig.10

III. TRANSFORMATION COMPUTATION IN COMPILING

The data of an IC pattern needs a large amount of graphic transformation computations. A computational algorithm of transformation using rectangulars as the basic graphic units is designed for ZB-761 which simplifies the program and makes it easier to read the pattern.

There are eight different kinds of transformations for a graph with sides parallel to the plane coordinates (the graph is composed of rectangulars). They are: translation, symmetrization with respect to the X axis, symmetrization with respect to the Y axis, symmetrization with respect to a center point, 90° clockwise rotation, 90° counterclockwise rotation, 90° clockwise rotation and symmetrization with respect to the Y axis and 90° counterclockwise rotation and symmetrization with respect to the Y axis. The latter two are nothing but the superpositions of the two transformations of the previous ones, thus any graphic transformation can be expressed in terms of six different kinds of transformations.

Symmetric axes, symmetric centers and rotation centers are usually considered in dealing with graphic transformations. When

a Cartesian coordinate system is transformed, the relation of a point's coordinates before the transformation and after the transformation can be expressed in the matrix form as:

$$[x' \ y' \ 1] = [x \ y \ 1] A$$

where x', y' are the coordinates after the transformation, x, y are the ones before the transformation, A is the following 3x3 matrix:

For translations,

$$A = P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{pmatrix}$$

where T_x, T_y are the amount of translation;

For symmetry transformations,

$$A = D = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

when $S_x = 1$, $S_y = -1$, D is an X-symmetrization matrix; when $S_x = -1$, $S_y = 1$, D is a Y-symmetrization matrix; when $S_x = S_y = -1$, D is a center-symmetrization matrix;

For rotations,

$$A = Z = \begin{pmatrix} 0 & -\sin\theta & 0 \\ \sin\theta & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

when $\theta = 90°$, Z represents a $90°$ clockwise rotation; when $\theta = -90°$, Z represents a $90°$ counterclockwise rotation.

This kind of transformation computation requires the user to find symmetric axes, symmetric centers and rotation centers. Moreover, since the operand is "points", it increases the amount

of work in compiling when these points are converted into rectangulars, the basic graphic units in the internal data storage.

ZB-761 gives a computational algorithm of transformation with rectangulers as the basic graphic units. With this computation the user does not have to find symmetric axes, symmetric centers and rotation centers. Instead, it is only required to find the transformation relation and the lower-left corner coordinates after the transformation, which is usually a very easy task. This reduces the amount of work in reading the sketch and avoids possible errors in finding symmetric axes, symmetric centers and rotation centers.

In this case, the transformation computation matrices are:

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ X_0 & Y_0 & 1 \end{bmatrix}, \quad D = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ \frac{1-S_x}{2}L_x & \frac{1-S_y}{2}L_y & 1 \end{bmatrix},$$

$$Z = \begin{bmatrix} 0 & -\sin\theta & 0 \\ \sin\theta & 0 & 0 \\ \frac{1-\sin\theta}{2}L_x & \frac{1+\sin\theta}{2}L_y & 1 \end{bmatrix}.$$

where X_0, Y_0 are the coordinates of the lower-left corner after the transformation; S_x, S_y and θ are as before; $L_x = L - \Delta x$, $L_y = W - \Delta y$, where L and W are the length and the width of the outer sides of the transformed graph, Δx and Δy are the length and the width of a rectangular inside of that transformed graph.

All the transformations must be postmultiplied by the P matrix as the last step.

These matrix representation of the transformation computation can be generalized to handle the data representing an array of

regularly arranged rectangulars.

IV. OPTIMIZATION OF THE OBJECTIVE PROGRAM

After compiling, the source program finally becomes sets of instructions and sets of data for the program controlled equipments to fabricate photomasks. These sets of instructions and sets of data are the objective program. The quality of the objective program directly affects the efficiency of the mask fabrication equipments and the quality of the photomasks. Hence it is necessary to optimize the objective program. We consider mostly the optimization of the cutting (drawing) program and the operation program of the pattern generator.

There are three parts in the optimization of the operation program of the pattern generator: the optimization of the lower platform path, the optimization of the slit path and the optimization of the upper platform path.

The pattern generator works in this way: a light beam from a source passing through an adjustable rectangular slit, projects a rectangular exposure region on the film fixed on the lower platform to fabricate intermediate-step photomasks. The lower platform can move along X and Y direction to change the position of the rectangular on the film and to make more rectangulars in different positions. Thus in order to fabricate an intermediate-step photomask, the rectangular slit must be changed frequently and the lower platform must be moved frequently. The problem of minimizing the total amount of distance the lower platform travels, and the number of changes the slit made, is the problem of optimizing the lower

platform path and the slit path of the program of the pattern generator.

Besides, for the pattern generator with a fixed pattern platform (upper platform), there are several premade pattern sets on the upper platform. When the upper platform is used to fabricate photomasks, the problem of minimizing the total amount of the distance the fixed platform moved is the optimization problem of the upper platform path.

The optimization of the lower platform path is most important among these three. We illustrate its significance with Fig.11.

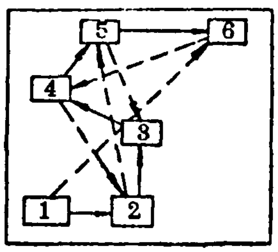


Fig.11

Suppose we want to generate six identical rectangulars as shown in Fig.11. In this case the slit remains unchanged, while the lower platform is moved to six prescribed positions to have the film exposed. The path indicated by the doted arrows (i.e. $1 \to 6 \to 4 \to 2 \to 5 \to 3$) is much longer than the path indicated by the solid arrows (i.e. $1 \to 2 \to 3 \to 4 \to 5 \to 6$). Obviously, short path means high efficiency in pattern generating. We can see from this example that the optimization problem of the lower platform is in fact the one of arranging the output order of subpatterns. In general, for n rectangulars to be generated, there are n! possibilities of different orderings. If we go through all the n! orderings to find an optimal one, we need to compute $(n/e)^n$ different schemes. When n is large (for CADP n is usually several dozen thousand or several hundred thousand), the amount of computation is tremendous and is impossible to implement.

ZB-761 uses the following optimization algorithm which includes two steps.

The first step is to generate the output subpatterns in the order of areas and to optimize the area order.

The entire pattern is divided into several areas. The amount of each area may be either fixed or varied according to the instructions of the user. For example, a pattern is divided into 16 areas which are numbered as in Fig.12 (a) or (b). The generator generates

4	5	12	13
3	6	11	14
2	7	10	15
1	8	9	16

Fig.12 (a)

16	15	14	13
9	10	11	12
8	7	6	5
1	2	3	4

Fig.12 (b)

subpatterns in the order of these areas, i.e. first it generates all the subpatterns in area 1, then generates those in area 2, and so on. Thus, if each area is viewed as a subpattern, the output order is optimal.

The second step is to arrange the generating order of the subpatterns in each area using "simplified postman's method". The idea of this method is to find the nearest subpattern with respect to the current one as the next working object, and to continue this process until all the subpatterns in the area are done. This method is only relatively optimal since it does not compute and compare all possible orderings. The rigorous optimal one requires the amount of work up to the order of $n!$, while the "simplified

postman's method" requires only that of n which reduces the computation time, simplifies the program, and exhibits a fairly good result in experiments.

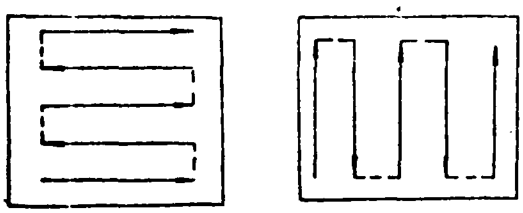
The optimization of the slit path and the upper platform path are simple and will not be discussed here.

The optimization of cutting (drawing) program is considered together with the problem of "elimination of extra lines". In the view of optimizing the drawing program, this problem is to arrange the output order of all the lines in the pattern so as to minimize the path of the pen.

ZB-761 solves this problem as follows. The lines in X direction and the lines in Y direction are drawn seperately. All the lines in X direction (or Y direction) are drawn first, then the lines in the other direction are drawn. This also makes it easier to change the direction of the cutting tool in the cutting program (if there are several cutting tools in the cutting machine, the problem of changing the direction of the cutting tool does not exist). The S-paths of drawing the lines in one direction are shown in Fig.13, in which (a) shows the path of drawing in X direction and (b) shows that in Y direction. Again this method is relatively optimal but not necessarily rigorously optimal. To find the rigorous optimal one requires again the amount of work up to the order of n!. However, a more important reason of using this method is to accommodate the computational algorithm of the "elimination of extra lines".

Since the pattern generator is regarded as the key equipment in the mask fabrication, the input language is designed so as to meet the requirements of the output of the generator. In other

words, it uses rectangulars as the basic graphic units. Thus there arises a problem of converting rectangular data into line segment data. For some graphs, "extra lines" appear if rectangulars are used as the basic graphic units. When the graph in Fig.14 is



(a) (b)

Fig.13

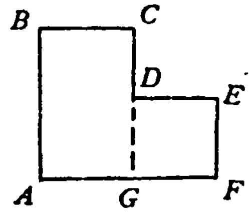


Fig.14

divided into two rectangulars, for example, the extra doted line DG appears. We call this kind of lines as "extra lines". When a pattern is generated in a pattern generator, the problem of extra lines will not appear even if the graph ABCDEF is obtained by putting two rectangulars together. However, an extra line will be drawn in drawing (cutting). These extra lines can be eliminated when the optimization of the drawing program is performed. One possible approach is to use the method of eliminating envelopes in graph theory which is internal storage consuming and time consuming. The other approach used in ZB-761 has the advantages of simplifying the program, saving the internal storage and reducing the computation time. The idea is as follows.

Let x_1, x_2 be the X coordinates of a horizontal side of a rectangular, $x_1 < x_2$, and x_3, x_4 be that of another one, $x_3 < x_4$, and they have the same Y coordinates. There are three cases regarding to the relation of the x_i's. The first case is shown in Fig.15,

$x_1<x_2<x_3<x_4$. The line segments $\overline{x_1 x_2}$ and $\overline{x_3 x_4}$ do not overlap. There is no extra line in X direction. The second case is shown in Fig.16, $x_1<x_3<x_2<x_4$. The line segments $\overline{x_1 x_2}$ and $\overline{x_3 x_4}$ overlap. The portion of overlaping, $\overline{x_3 x_2}$, is the so called "extra line". In this case, the output line segments are reduced to $\overline{x_1 x_3}$ and $\overline{x_2 x_4}$. $\overline{x_3 x_2}$ is eliminated. A degenerated case of this happens when $x_3=x_2$, the extra line is degenerated into a point. The output line segment is $\overline{x_1 x_4}$ after eliminating this "extra point" (Fig.17). The third case is shown in Fig.18, $x_3<x_1<x_2<x_4$. The line segment $\overline{x_1 x_2}$ is contained in $\overline{x_3 x_4}$. In this case the extra line is $\overline{x_1 x_2}$. The output line segments are reduced to $\overline{x_3 x_1}$ and $\overline{x_2 x_4}$. When $x_3=x_1$ or $x_2=x_4$ (as shown in Fig.19), the output line segment is only $\overline{x_2 x_4}$ or $\overline{x_3 x_1}$. When $x_3=x_1$, and $x_2=x_4$, the line segments $\overline{x_1 x_2}$ and $\overline{x_3 x_4}$ coincide each other. They are all extra lines and there is no output line segment (as shown in Fig.20).

The above three cases are combined into the same computational algorithm. The coordinates of these four points are arranged in

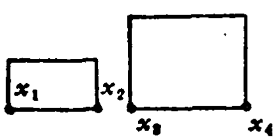


Fig.15

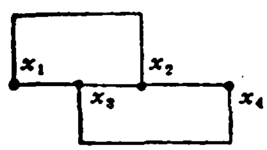


Fig.16

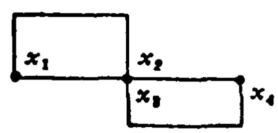


Fig.17

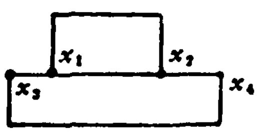


Fig.18

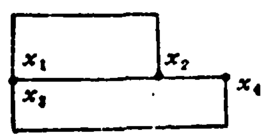


Fig.19

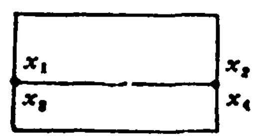


Fig.20

value increasing order. If there are points having the same coordinates they should be eliminated. Then the remaining points are coupled from the left to the right to form the output line segments.

This idea can be generalized to any number of points. The algorithm is easy to program. After eliminating the extra lines in X direction, the same algorithm can be applied to Y direction to eliminate the extra lines in that direction. The output order of line segments after eliminating the extra lines is the above mentioned optimal order.

V. RESULT CHECKING AND MAN-MACHINE CONVERSATION

The photomask fabrication program is essentially a program for processing data. It needs to handle more data than for scientific calculation. Hence, an important problem is how to check these data results.

Several means for data result checking are considered as follows.

(1) Syntax error and semantic error checking in compiling, which can indicate these kind of errors.

(2) Technological specification checking. The user may prescribe some technological specifications such as the width, the interspace and the registering tolerance of metal interconnection lines and diffusion regions by means of technological specification declaration statements, and then use the program to check whether the pattern data meet these specifications. This can detect the data errors that are correct in the sense of syntax.

(3) Stepwise and blockwise checking. At each step of the program execution the current data stored in the internal storage can be printed out for check by typing in instructions from the keyboard. The source program can also be tested and checked block by block because of the block structure.

(4) Checking by means of drawing and displaying. Parts of the pattern or the entire pattern can be drawn using the drawing machine or be displayed using the pattern display device. Usually the display device is used for area checking while the drawing machine is used for the entire pattern checking. Since drawing is well visualized and it is possible to draw several patterns of different layers in the same drawing using different colors for registering tolerance checking, it is usually used as the means of the final check.

(5) Checking of the program controlled paper tape of the pattern generator. In case that the pattern generator is operated by being disconnected with the computer and using a program controlled paper tape as the input, it is necessary to check this paper tape to prevent the data errors on the tape caused by malfunctions of the computer or the peripheral equipments.

The method used in ZB-761 is to re-input the tape into the computer and to have it checked by a special checking program which can either "recover" the data on the tape into the original pattern data and then output them by printing, displaying or drawing according to the user's instruction or compare them with the data stored in the internal storage and check them one by one. The "recover" used here means to recover the tape data into the data

similar to the final results.

In one word, pattern data checking possesses an important position in the mask fabrication software since it is a vital means of improving fabrication efficiency, reducing errors and avoiding doing the same work over again.

Users usually wish to check and modify the pattern data in each step of the program execution. To meet this requirement a set of keyboard instructions is set up in ZB-761 which controls the execution of the program in each step by means of man-machine conversation. Besides, the lightpen display device can be used to check and modify the pattern and to display the conversation.

VI. THE IDEA AND PREPARATION OF DESIGN AUTOMATIZATION

ZB-761 software is only an aid in mask design and fabrication. Manual work such as drawing the pattern sketch, reading out necessary data (although the amount of the data read can be reduced by improving the software) and writing the source program is still required. However, the design automatization has been considered and some preparation has been done in designing ZB-761.

The following aspects are considered for the pattern design automatization.

(1) Set up the unit circuit library. This is both a software matter and a technological one. Assembly and layout automation are based on this issue. For the consideration of the software management, classifiing, indexing and protection of the library should be taken into account. And of course the data structure of the library and of these unit circuits must be designed. We made

a preliminary attempt to tackle these problems, that is, we considered these data structure and the indexing methods. For the consideration of the technology, the subpatterns of various unit circuits should be standardized and be normalized to file.

(2) Automatic design subroutines. This method can be applied to several kinds of regular unit circuits such as ROM's and decoders. This is in fact another kind of unit circuit library. What is stored in the above mentioned unit circuit library is the subpattern data while it is the standard subroutines for forming corresponding subpatterns in this case. The subroutines for forming MOS's and decoders are set up in ZB-761. Thus it is only required to give some technological specifications and circuit information in forming these subpatterns instead of reading the sketch.

(3) "PLA" automatic design. PLA is the abbreviation for programmable logic array. ZB-761 can handle regular patterns such as registers and shift registers very easily. In case of irregular circuits, however, more sketch reading is required. Using PLA, some irregular circuits can be decomposed into "AND" matrix circuits and "OR" matrix circuits whose patterns are regular. Thus the patterns of irregular circuits can be transformed into regular array patterns which can be designed automatically by software. We call this design method as "PLA" automatic design.

(4) Assembly using man-machine conversation. With the above means, the assembly can be done by assembling subpatterns of unit circuits into larger pattern blocks of functional circuits and then into the entire pattern using man-machine conversation. By this way the amount of manual work in drawing the sketch and reading

the data can again be reduced. Also, check and modification can be performed easily during the assembly.

(5) Automatic layout design. Based on the above discussion, the research on automatic layout design program can begin if conditions allow. This leaves much work to be done so far.

Since 1976 when ZB-761 was put into use, several dozens of the intermediate-step photomasks have been made, of which there are 12 bits random access storages, 13 bits MOS ROM's, and 10 bits MOS dynamic shift registers, etc. Some modification and augmentation has also been made. Because of the limitation of our knowledge, there are still some imperfections in the software that need to be improved further.

The personnel participated in the research of the software is as follows: Cai Da-yong, Wu Qi-ming, He Xi-zhang , Zhang Zhu-ping, Gao Yong-lin, Zhu Ming-xue and some current graduates majoring in computer programming. The personnel of Group 209, Institute of Semiconductor, Academia Sinica, offered us a great help in the research. We also appreciate the coordination of Sun Jia-guang and Cheng Yu-lin in software testing.

REFERENCE

[1]. Cai Da-yong & Hong Xian-long, A language for CAD photomask --- ZB-761, Transactions of Qinhua Univ. vol.1,1978.

A Software for CAD Photomask-ZB-761

Hong Xian-long Department of Electronic Engineering

Abstract

As a part of the LSI CAD, a software for CAD photomask-ZB-761 was designed and put into use on a DJS-130 computer in 1976. Since then several dozens of IC photomasks have been made.

LMED

8